Automatic Verification of Conformance of Firewall Configurations to Security Policies

Nihel Ben Youssef
Higher School of Communication of Tunis, Tunisia
nihel.benyoussef@gmail.com

Adel Bouhoula
Higher School of Communication of Tunis, Tunisia
adel.bouhoula@supcom.rnu.tn

Florent Jacquemard
INRIA & LSV (CNRS/ENS Cachan), France
florent.jacquemard@lsv.ens-cachan.fr

Abstract

The configuration of firewalls is highly error prone and automated solution are needed in order to analyze its correctness. We propose a formal and automatic method for checking whether a firewall reacts correctly wrt a security policy given in an high level declarative language. When errors are detected, some feedback is returned to the user in order to correct the firewall configuration. Furthermore, the procedure verifies that no conflicts exist within the security policy. We show that our method is both correct and complete. Finally, it has been implemented in a prototype of verifier based on a satisfiability solver modulo theories (SMT). Experiment conducted on relevant case studies demonstrate the efficiency and scalability of the approach.

Keywords: *security, firewall configuration, security policy, formal verification, SMT solver.*

1. Introduction

Securing network flows is a crucial and difficult task. Some specific technical knowledge and clear ideas about the global security requirement to be establish are required, but are still insufficient in order to ensure the correctness of a security configuration. One of the main reasons is the difference in semantics between, on the one hand, the *security policies* (SP) used to express global security requirements, and on the other hand the *firewall configuration* files (FC). The former are generally specified in a high level declarative language and easy to understand, whereas the latter are low-level files subject to special configuration constraints in order to ensure an efficient real time processing by specific devices. One of these constraints is in particular that the filtering rules of a FC file are treated in the order in which they are read in the configuration file, in a *switch-case* fashion. For instance, if two filtering rules associate different actions to the same flow type, then only the rule with the lower order is really applied. This is in contrast with the SP, which is a set of rules considered without order. In this case, the action taken, for the flow under consideration, is the one of the non executed rule.

Several methods have been proposed [12, 2, 3, 1] for the detection of inter-rule conflicts in FC. These work are limited to the problem conflict avoidance, and do not consider the more general problem of verifying whether a firewall reacts correctly wrt a given SP. Solutions are studied in [9], [6], and [10] for the analysis of firewalls' behavior. These methods require some final user interactions by sending queries through a verification tool. Such manual solution can be tedious when checking discrepencies wrt complicated security requirements. In [4] and [8] the author address the problem of automatic verification by providing automatic translation tool of the security requirements (SP), specified in a high level language, into a set of ordered filtering rule (i.e. a FC). Therefore, these methods can handle the whole problem of conformance of FC to SP, but the validity of the compilation itself has not been proven. In particular, the FC rules obtained may be in conflict.

In this paper, we propose an automatic and generic method for checking whether a firewall is well configured wrt a security policy, given in an expressive enough declarative language. Furthermore, the proposed method ensures conflicts avoidance within the SP that we aim to establish and returns key elements for the correction of a flawed FC. Our method has been implemented as a prototype which can be used either in order to validate an existing FC wrt a given SP or downstream of a compiler of SP. It can also be used in order to assist the updates of FC, as conflicts may be created by the addition or deletion of filtering rules.

The remainder of this paper is organized as follows. Section 2 settles the definition of the problems addressed in the

paper, in particular the properties of soundness and completeness of a FC wrt SP. In Section 3, we present an inference system introducing the proposed method and prove its correctness and completeness. Section 4 elaborates our the verification procedure. Finally, in Section 5 we present an automatic verification tool that we have implemented based on our method and some experiments on a case study.

2. Soundness and Completeness Properties

The main goal of this work consists in checking whether a FC is sound and complete wrt a given SP. In this section, we define formally these notions.

We consider a finite *domain* $\mathcal{P}$ containing all the headers of packets possibly incoming to or outgoing from a network. A *firewall configuration* (FC) is a finite sequence of *filtering rules* of the form $FC = (r_i \Rightarrow A_i)_{0 \leq i < n}$. Each precondition r_i of a rule defines a filter for packets of $\mathcal{P}$. The structure of r_i is described later in Section 5. Before, we just consider a function dom mapping each r_i into the subset of $\mathcal{P}$ of filtered packets. Each right member A_i of a rule of FC is an *action* defining the behaviour of the firewall on filtered packets: $A_i \in \{accept, deny\}$. This model describes a generic form of FC which are used by most firewall products such as CISCO, *Access Control List*, `IPTABLES`, `IPCHAINS` and *Check Point Firewall*...

A *security policy* (SP) is a set SP of formulae defining whether packets are accepted or denied. The nature of the formulae is described in Section 4. In Section 3, we only consider the definition domain of SP, partitioned into $dom(SP) = \bigcup_{A \in \{accept, deny\}} SP_A$. SP is called *consistent* if $SP_{accept} \cap SP_{deny} = \emptyset$.

A FC is *sound* wrt a SP if the action undertaken by the firewall for each forwarding packet, (i.e. the action of the first filtering rule matching the packet) is the same as the one defined by the SP.

Definition 1 (soundness) *FC is sound wrt SP iff for all $p \in \mathcal{P}$, if there exists a rule $r_i \Rightarrow A_i$ in FC such that $p \in dom(r_i)$ and for all $j < i$, $p \notin dom(r_j)$ then $p \in SP_{A_i}$.*

A FC is *complete* wrt a SP if the action defined by the SP for each packet p is really undertaken by the Firewall.

Definition 2 (completeness) *FC is complete wrt SP iff for all $p \in \mathcal{P}$ and $A \in \{accept, deny\}$, if $p \in SP_A$ then there exists a rule $r_i \Rightarrow A$ in FC such that $p \in dom(r_i) \setminus \bigcup_{j<i} dom(r_j)$.*

3 Inference System

We propose in this section necessary and sufficient condition for for the simultaneous verification of the properties of soundness and completeness of a FC wrt a SP. The conditions are presented in an inference system shown in Figure 1. The rules of the system in Figure 1 apply to couples (FC, D) whose first component FC is a sequence of filtering rules and whose second component D is a subset of $\mathcal{P}$. This latter subset represents the accumulation of the sets of packets filtered by the rules of FC processed so far.

$$\text{recurcall} \quad \frac{((r \Rightarrow A); FC), D}{FC, D \cup dom(r)} \quad \text{if } dom(r) \setminus D \subseteq SP_A$$

$$\text{success} \quad \frac{\emptyset, D}{\mathsf{success}} \quad \text{if } D \supseteq dom(SP)$$

$$\text{failure} \quad \frac{FC, D}{\mathsf{fail}(fst(FC), D)} \quad \text{if no other rule applies}$$

Figure 1. Inference System

We write $C \vdash_{SP} C'$ is C' is obtained from C by application of an inference rule of Figure 1 (note that C' may be a couple as above or one of success or fail) and we denote by $\vdash^*_{SP}$ the reflexive and transitive closure of $\vdash_{SP}$.

recurcall is the main inference rule. It deals with the first filtering rule $r_i \Rightarrow A$ of the FC given in the couple. The condition for the application of this rule is that the set of packets $dom(r_i)$ filtered by this rule and not handled by the previous rules (i.e. not in D) would follow the same action A according to the the security policy. Hence, successful repeated applications of recurcall ensures the soundness of the FC wrt the SP.

The success rule is applied under two conditions. First, recurcall must have been used successfully until all filtering rules have been processed (in this case the first component FC of the couple is empty). Second, the domain global security policy must be included in D. This latter condition ensures that all the packets treated by the security policy are also handled by the firewall configuration (completeness of FC).

There are two cases for the application of failure. In the first case, failure is applied to a couple (FC, D) where FC is not empty. It means that recurcall has failed on this couple and hence that the FC is not sound wrt the SP. In this case, failure returns the first filtering rule of FC as a example of rule which is not correct, in order to provide help to the user in order to correct the FC. In the second case, failure is applied to $(\emptyset, D)$. It means that success has failed on this couple and that the FC is not complete wrt the SP. In this case, D is returned and can be used in order to identify packets handled by the SP and not by the FC.

Let us now prove that the inference system of Figure 1 is correct and complete. From now on, we assume given a FC $FC = r_0 \Rightarrow A_0, \ldots, r_{n-1} \Rightarrow A_{n-1}$ with $n > 0$.

In the correctness theorem below, we assume that SP is consistent. We shall present in the next sections a method for checking this property.

Theorem 1 (correctness) *Assume that the security policy SP is consistent. If $(FC, \emptyset) \vdash^*_{SP}$ success then the firewall configuration FC is sound and complete wrt SP.*

Proof. If $(FC, \emptyset) \vdash^*_{SP}$ success then we have $(FC, \emptyset) \vdash_{SP} (FC_1, D_1) \vdash_{SP} \ldots \vdash_{SP} (FC_n, D_n) \vdash_{SP}$ success, where $FC_n = \emptyset$, all the steps but the last one are recurcall and $dom(SP) \subseteq D_n$. We can easily show by induction on that for all $1 \leq i \leq n$, $D_i = \bigcup_{j<i} dom(r_j)$. Let $D_0 = \emptyset$.

Assume that there exists $p \in \mathcal{P}$ and $r_i \Rightarrow A_i$ in FC $(i < n)$ such that $p \in dom(r_i) \setminus \bigcup_{j<i} dom(r_j)$. It follows that $p \in dom(r_i) \setminus D_i$, and, by the condition of recurcall that $p \in SP_{A_i}$. Hence FC is sound wrt SP.

Let $A \in \{accept, deny\}$ and $p \in SP_A$. By the condition of the inference rule success, $p \in D_n = \bigcup_{j<i} dom(r_j)$. Let i be the smallest integer k such that $p \in dom(r_k)$. It means that $p \in dom(r_i) \setminus \bigcup_{j<i} dom(r_j)$. As above, it follows that $p \in SP_{A_i}$, and hence that $A_i = A$, by the hypothesis that SP is consistent. Therefore, FC is complete wrt SP. □

Theorem 2 (completeness) *If the firewall configuration FC is sound and complete wrt the security policy SP then $(FC, \emptyset) \vdash^*_{SP}$ success.*

Proof. Assume that FC it is sound and complete wrt SP. The soundness implies that for all $i < n$ and all packet $p \in dom(r_i) \setminus \bigcup_{j<i} dom(r_j)$, $p \in SP_{A_i}$. It follows that $(FC, \emptyset) \vdash_{SP} (FC_1, D_1) \vdash_{SP} \ldots \vdash_{SP} (FC_n, D_n)$ with $D_i = \bigcup_{j<i} dom(r_j)$ for all $i \leq n$ and $FC_n = \emptyset$, by application of the inference recurcall. Moreover, the completeness of FC implies that every $p \in dom(SP)$ also belongs to D_n. Hence $(FC_n, D_n) \vdash_{SP}$ success, and altogether $(FC, \emptyset) \vdash^*_S P$success. □

Theorem 3 (soudness of failure) *If $(FC, \emptyset) \vdash^*_{SP}$ fail then the firewall configuration FC is not sound or not complete wrt the security policy SP.*

Proof. Either we can apply iteratively the recurcall rule starting with $(FC, \emptyset)$, until we obtain $(\emptyset, \bigcup_{j<n} dom(r_j))$, or one application of the recurcall rule fails. In the latter case, there exists $i < n$ such that $dom(r_i) \setminus \bigcup_{j<i} dom(r_j) \nsubseteq SP_{A_i}$. Therefore, there exists $p \in \mathcal{P}$ such that $p \in dom(r_i) \setminus \bigcup_{j<i} dom(r_j)$ and $p \notin SP_{A_i}$. It follows that FC is not sound wrt the security policy SP.

If $(FC, \emptyset) \vdash^*_S P(\emptyset, \bigcup_{j<n} dom(r_j))$ using recurcall but the application of the success rule to the last couple fails, then there exists $A \in \{accept, deny\}$ and $p \in SP_A$ such that $p \notin \bigcup_{j<n} dom(r_j)$. It follows that FC is not complete wrt the security policy SP. □

$$\left(\begin{array}{lcll} c_1 & \Rightarrow & accept & \mid e_1 \\ c_2 & \Rightarrow & deny & \\ c_3 & \Rightarrow & accept & \\ c_4 & \Rightarrow & deny & \mid e_4 \end{array}\right)$$

Figure 2. A Security Policy

Since the application of the inferences to $(SP, \emptyset)$ always terminates, and the outcome can only be success or fail, it follows immediately from the Theorem 1 that if the firewall configuration FC is not sound or not complete wrt the security policy SP, then $(FC, \emptyset) \vdash^*_{SP}$ fail (completeness of failure).

To summarize the above results, we have the following sufficient and necessary conditions for

soundness: $\forall i < n$, $dom(r_i) \setminus \bigcup_{j<i} dom(r_j) \subseteq SP_{A_i}$,

completeness: soundness and $dom(SP) \subseteq \bigcup_{i<n} dom(r_i)$.

4 Verification Procedures

We describe in this section how to express the conditions for the above inference system of Section 3 as satisfiability problems for propositional formulae.

We consider a security policy which is presented as a finite set of *directives*: $SP = \{c_i \Rightarrow A_i | e_i \mid 1 \leq i \leq m\}$. Each directive can be simple or complex. A simple directive describes whether a traffic destinated to one or more services that are required by one or more sources and given by one or more destinations (as described by the condition c_i) must be accepted or refused (according to $A_i \in \{accept, deny\}$). A complex directive is basicly a simple directive but considering some exceptions defined in e_i. The following examples are respectively simple and complex directives.

- *A sub network LAN_A has the right to access to the web service provided by a machine B located in the sub network LAN_B.*
- *A sub network LAN_A, except the machine A', has the right to access to the web service provided by a machine B located in the sub network LAN_B.*

The condition of the above second directive refers to the flow between the source LAN_A and the destination B, the directive action is $accept$, and its exception is the source machine A'. The conditions and the exceptions are expressed as conjunctions of logic propositions describing

the following main fields: source(s), destination(s) and service(s).

4.1 Security Policy Consistency

As explained in Section 3, the SP must be consistent in order to ensure the correctness of the verification method. Hence in a first step, we shall check that the domains of accepted and denied packets, SP_{accept} and SP_{deny}, do not overlap. The set of accepted packets SP_{accept}, is the union of one set for each directive. Each set represents either the domain of a simple directive, if the action of the corresponding condition is $accept$, or the domain of a complex directive's exception, if its action is $deny$ or else the difference between the domains of the condition and the exception of a complex directive if its main action is $accept$. The set of denied packets is definied symmetrically. We express the above sets by propositional formulas ϕ_a and ϕ_d defining respectivelly the domains of accepted and denied packets. In the case of the SP of Figure 2, their definitions are

$$\begin{array}{rcl} \phi_a & ::= & (c_1 \wedge \neg e_1) \vee c_3 \vee e_4 \\ \phi_d & ::= & e_1 \vee c_2 \vee (c_4 \wedge \neg e_4) \end{array}$$

In order to ensure that the domains do not overlap, we prove the insatisfiability of the following propositional formula $\phi_a \wedge \phi_d$. It ensures that no packet satisfies simultaneously ϕ_a and ϕ_d.

4.2 Soundness Verification

Our goal is to check whether a firewall configuration $FC = \{r_i \Rightarrow A'_i \mid 0 \leq i < n\}$ is sound wrt the above security policy SP.

According to Section 3, the condition for the soundness of FC wrt SP is that, for each filtering rule $r_i \Rightarrow A'_i$, the domain $dom(r_i)$ minus the accumulated domains of the previous rules, is included in the set of packets that should undergo, according to SP, the same action as A'_i of the filtering rule. The soundness condition is equivalent to the insatisfiability of all the following propositional formulae, for $0 \leq i < n$ (expressing the negation of the soundness condition in Section 3): $\phi_i \wedge \neg\phi_{A_i}$ where $\phi_i ::= \phi_{r_i} \wedge \neg(\vee_{j=1}^{i-1}\phi_{r_j})$ and ϕ_{A_i} is defined as above.

4.3 Completeness Verification

As explained in Section 3, in order to verify the completeness property of a firewall configuration, we check that the domain of the security policy SP is included in that of the firewall configuration FC. Theses domains are made of the unions of the respective domains of filtering rules and security directives, as expressed by the following formulae.

$$\phi_{SP} ::= \bigvee_{i=1}^{m} c_i \qquad \phi_{FC} ::= \bigvee_{i=0}^{n-1} r_i$$

The completeness condition of Section 3 is the insatisfiability of of the following formula: $\phi_{SP} \wedge \neg\phi_{FC}$.

5 Automatic Verification tool

The above presentation of the conditions of Section 3 permits the automation of the verification of soudness and completeness of FC wrt SP. In fact, in the previous section, these properties are expressed as satisfiability problems.

We have used a recent satisfiability solver modulo theories, Yices [5], in order to describe the different inputs and to automate the verification process. Yices provides different additional functions, compared to simple satisfiability solvers. These functions are based on theories like those of arrays, list structures and bit vectors.

The first input of our verification tool is a set of firewall rules. Each rule is defined by a priority order and composed of the following main fields: the source, the destination, the protocol and the port. The source and destination fields correspond to one or more machines identified by an IPv4 address and a mask coded both on 4 bytes, each bit being represented by a proposition variable. For better readability, we consider in this section the IP coding on 4 bits. For example, the following expression written in Yices syntax refers to the third filtering rule concerning `UDP` or `TCP` flow, coming from the source network `1.0.0.0/2` and reaching the network `1.1.0.0/3` for a destination port belonging to the subrange $[20 - 60]$.

```
(define r::(->intbool))(assert(=(r3)(andips4
(notips3)ipd4ipd3(notipd2)(>=port20)(<=port60)
(or(=protocoltcp)(=protocoludp))))).
```

In order to illustrate the verification procedure proposed, we have chosen to apply our method to a case study of a corporate network represented in Figure 3. The network is divided into three zones delineated by branches of a firewall whose initial configuration FC corresponds to the rules in Table 1. The security policy SP that should be respected contains the following directives.

sp_1 : `net_1` has the right to access to `net_2`.

sp_2 : The machine `admin` has the right to access to the FTP service provided by the FTP server belonging to `net_3`.

sp_3 : `net_1` has not the right to access to the DNS service provided by the DNS server belonging to `net_2`.

In the following, we note that d_i and d'_i are respectively the condition and the exception of the security directive sp_i.

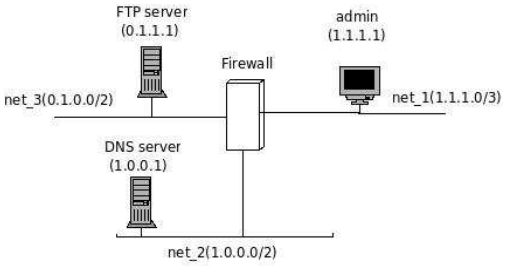

Figure 3. Network Architecture Example

	src_adr	*dst_adr*	*protocol*	*dst_port*	*action*
r_1	1.1.1.1	0.1.1.1	21	tcp	accept
r_2	1.1.1.0	1.0.0.1	53	udp	deny
r_3	1.1.1.0/3	1.0.0.0/3	*	*	accept

Table 1. Firewall Configuration to be Verified

Our goal is to verify that the configuration FC is conform to the security policy SP by checking the soundness and the completeness properties. But first, we must verify that the security policy is consistent.

5.1 Security Policy Consistency Verification

After we specified the security policy SP in Yices syntax, checking the consistency of SP amounts to check satisfiability. We have obtained the satisfiability result displayed in Figure 4.

Note that the Yices option `e` (evidence) displays a model if the logic expression is satifiable. In our case, this means that the security policy is not consistent. An analysis of the model obtained shows that the directives sp_1 and sp_3 have contradictory decisions. It concerns on the flow coming from a machine belonging to `net_1` and destinated the DNS service of the DNS server `1.0.0.1`. Indeed, the third directive indicates that the network `net_1` has not the right to access to the DNS server of `net_2` while the first directive permits it. One possible correction would be to replace the two directives in conflict by the following single complex directive.

```
# ./yices -e verif_coherence.ys

loading security_policy.ys... done
sat
(= ips4 true)
(= ips3 true)
(= ips2 true)
(= ipd4 true)
(= ipd3 false)
(= ips1 true)
(= ipd2 false)
(= ipd1 true)
(= port 53)
(= protocol udp)
(= (d 1) true)
(= (d 2) false)
(= (d 3) true)
```

Figure 4. Coherence verification

```
# ./yices -e soundness.ys
loading security_policy.ys... done
loading Firewall_rules.ys... done
sat
(= ips4 true)
(= ips3 true)
(= ips2 true)
(= ipd4 true)
(= ipd3 false)
(= ips1 true)
(= ipd2 false)
(= ipd1 true)
(= port 53)
(= protocol udp)
(= (d 1) true)
(= (d 2) false)
(= (d' 1) true)
(= (r 1) false)
(= (r 2) false)
(= (r 3) true)
```

Figure 5. Soundness verification

sp_1 : `net_1` has the right to access to `net_2` except the DNS server (DNS service).

5.2 Soundness Verification

Once ensured that the security policy is consistent, we proceed to the verification of the firewall configuration soundness, using the formulae of Section 4, which express the conditions of Section 3. The satisfiability result obtained is displayed in Figure 5. The outcome shows that the firewall configuration FC is not sound wrt the security policy SP, i.e. that there exists some packets that will undergo an action different of that imposed by the security policy. It indicates also that r_3 is the first rule that causes this discrepancy precisely with the exception of directive sp_1. Indeed, the model returned corresponds to a packet accepted by the firewall through the rule r_3 while it should be refused according to the first directive of the security policy. This conflict can be resolved by adding a rule immediately preceding the rule r_3, which allows to implement the first directive exception precised by the given model. Table 2 presents this modification.

	src_adr	*dst_adr*	*protocol*	*dst_port*	*action*
r_1	1.1.1.1	0.1.1.1	21	tcp	accept
r_2	1.1.1.0	1.0.0.1	53	udp	deny
r_3	1.1.1.1	1.0.0.1	53	udp	deny
r_4	1.1.1.0/3	1.0.0.0/3	*	*	accept

Table 2. A Sound Firewall Configuration

5.3 Completeness Verification

After that the soundness property has been established, we proceed to the verification of the completeness of the firewall configuration. We obtained the satisfiability result displayed in Figure 6.

According to this outcome, the configuration FC is not complete wrt the security policy SP: some packets handled by the security policy are not treated by the filtering rules. Indeed, the rule r_4 addresses only a subnetwork of `net_2`.

```
# ./yices -e verif_completeness.ys
loading security_policy.ys... done
loading Firewall_rules.ys... done
sat
(= ips4 true)
(= ips3 true)
(= ips2 true)
(= ipd4 true)
(= ipd3 false)
(= ips1 false)
(= ipd2 true)
(= ipd1 false)
(= port 54)
(= protocol udp)
(= (d 1) true)
(= (d 2) false)
(= (d' 1) false)
(= (r 1) false)
(= (r 2) false)
(= (r 3) false)
(= (r 4) false)
```

Figure 6. Completeness verification

The packet corresponding to the model returned belongs to another part of `net_2`, which is untreated. One possible solution would be to change, as shown in Table 3, the mask used in the destination address of the rule r_4 to consider all the first directive domain.

	src_adr	*dst_adr*	*protocol*	*dst_port*	*action*
r_1	1.1.1.1	0.1.1.1	21	tcp	accept
r_2	1.1.1.0	1.0.0.1	53	udp	deny
r_3	1.1.1.1	1.0.0.1	53	udp	deny
r_4	1.1.1.0/3	1.0.0.0/2	*	*	accept

Table 3. A sound and complete FC

We note that YICES validated the three properties after the modifications taken in sections *5.1*, *5.6* and *5.3* by displaying in each case an insatisfiability result.

5.4 Experimental Results

In order to better assess the performance of the Yices tool for checking satisfiability of propositional formulas, we consider time treatement factor that we review by varying the different parameter sizes provided by the solver. In overall terms, we consider the average processing time of the three verification procedures (securtity policy consistency, firewall configuration soundness and completeness) by varying the total number of Boolean variables used. We note that we treat real field values such IPv4 address and [1-65535] port range. To perform this study, we used the option `st` of Yices to collect statistical information. The experimental tests were conducted on an Intel Dual core 1.6 GHz with 1 Gbyte of RAM.

$bool\_var\_nb$	$processing\_time(s)$
20	0.004
65	0,020
79	0,028
155	0,09
179	0,21

Table 4. Processing time evaluation

The experimental results are summarized in Table 4. The tool processing time obtained shows that our tool performed efficiently on the case studies.

6 Conclusion

In this paper, we propose a formal method for certifying automatically that a firewall configuration is sound and complete wrt a security policy. Otherwise, the method provides key information helping users to correct configuration errors. Our method permits also to prove the consistency of a security policy which is a necessary condition for the verification procedure. Finally, our method has been implemented using the satisfiability solver Yices. The experimental results obtained are very promising.

As further work, we are considering to extend the approach in order to handle stateful firewall configurations.

References

[1] T. Abbes, A. Bouhoula, and M. Rusinowitch. Inference system for detecting firewall filtering rules anomalies. In *Proc. of the 23rd annual ACM Symp. on Applied Computing*, 2008.

[2] E. Al-Shaer and H. Hamed. Firewall policy advisor for anomaly detection and rule editing. In *IEEE/IFIP Integrated Management, IM'2003*, 2003.

[3] M. Benelbahri and A. Bouhoula. Tuple based approach for anomalies detection within firewall filtering rules. In *12th IEEE Symp. on Computers and Communications*, 2007.

[4] F. Cupens, N. Cuppens-Boulahia, T. Sans, and A. Miege. A formal approach to specify and deploy a network security policy. In *In Second Workshop on Formal Aspects in Security and Trust*, pages 203-218, 2004.

[5] B. Dutertre and L. Moura. The yices smt solver. Available at `http://yices.csl.sri.com/tool-paper.pdf`, 2006.

[6] P. Eronen and J. Zitting. An expert system for analyzing firewall rules. In *Proc. of 6th Nordic Workshop on Secure IT Systems*, 2001.

[7] M. Gouda and A. X. Liu. Firewall design: consistency, completeness and compactness. In *In Proc. of the 24th IEEE Int. Conf. on Distributed Computing Systems*, 2004.

[8] H. Hamdi, M. Mosbah, and A. Bouhoula. A domain specific language for securing distributed systems. In *Second Int. Conf. on Systems and Networks Communications*, 2007.

[9] S. Hazelhusrt. Algorithms for analyzing firewall and router access lists. TR, Univ. of the Witwatersrand, 1999.

[10] A. X. Lui, M. Gouda, H. Ma, and A. Ngu. Firewall queries. In *Proc. of the 8th Int. Conf. on Principles of Distributed Systems*, pages 197-212, 2004.

[11] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proc. of the 2000 IEEE Symp. on Security and Privacy*, pages 14–17, 2000.

[12] C. Pornavalai and T. Chomsiri. Firewall policy analyzing by relational algebra. In *The 2004 Int. Technical Conf. on Circuits/Systems, Computers and Communications*, 2004.